

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-202720

(43)Date of publication of application : 19.07.2002

(51)Int.Cl.

G09C 1/00

G06F 1/00

G06F 9/46

G06F 12/14

(21)Application number : 2000-402672

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 28.12.2000

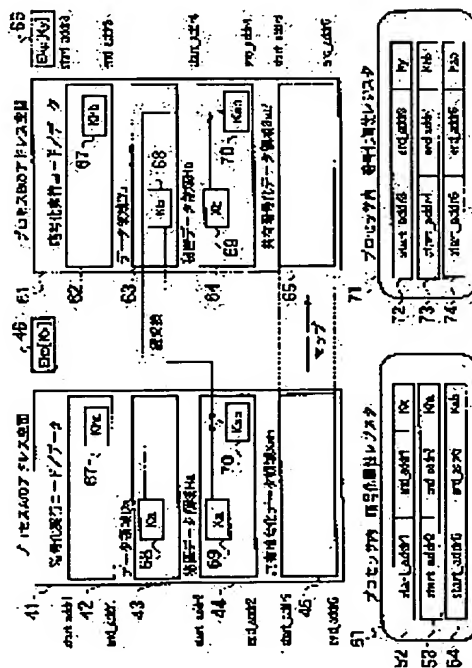
(72)Inventor : TERAMOTO KEIICHI
HASHIMOTO MIKIO
SHIRAKAWA KENJI
OZAKI SATORU
FUJIMOTO KENSAKU

(54) METHOD FOR SHARING ENCIPHERED DATA AREA AMONG PROCESSES IN A TAMPER-RESISTANT PROCESSOR

(57)Abstract:

PROBLEM TO BE SOLVED: To share a secret program or a secret data among a plurality of processes on a tamper-resistant microprocessor for supporting an environment for executing a multi-task program.

SOLUTION: An executing mode of a processor, having the functions of ciphering/deciphering a program and data, is shifted to a ciphering instruction executing mode, to make the processor generate respective own secret data areas in each process space of two processes. Further, each process is made to generate a different pair of keys from each other used for key exchange, and the key exchange is made to perform between the two processes. Based on the key exchange, each process is made to calculate the common key. The common key and the data used in the process of the key exchange are stored in the secret data area of each process. One of the processes is made to generate a shared ciphering area to be shared between the two processes, and the other process is made to map this shared ciphering area in own process space. Finally, the shared ciphering area and the common key are made to be related with each other and stored in a ciphered attribute register inside of the processor.



BEST AVAILABLE COPY

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号

特開2002-202720

(P2002-202720A)

(43)公開日 平成14年7月19日(2002.7.19)

(51)Int.Cl. ⁷	識別記号	F I	テーマコード(参考)
G 0 9 C 1/00	6 1 0	G 0 9 C 1/00	6 1 0 Z 5 B 0 1 7
G 0 6 F 1/00		G 0 6 F 9/46	3 4 0 B 5 B 0 7 6
	9/46	12/14	3 2 0 B 5 B 0 9 8
	12/14	9/06	6 6 0 L 5 J 1 0 4

審査請求 未請求 請求項の数7 O L (全 17 頁)

(21)出願番号 特願2000-402672(P2000-402672)

(22)出願日 平成12年12月28日(2000.12.28)

(71)出願人 000003078

株式会社東芝

東京都港区芝浦一丁目1番1号

(72)発明者 寺本 圭一

神奈川県川崎市幸区小向東芝町1番地 株

式会社東芝研究開発センター内

(72)発明者 橋本 幹生

神奈川県川崎市幸区小向東芝町1番地 株

式会社東芝研究開発センター内

(74)代理人 100083806

弁理士 三好 秀和 (外7名)

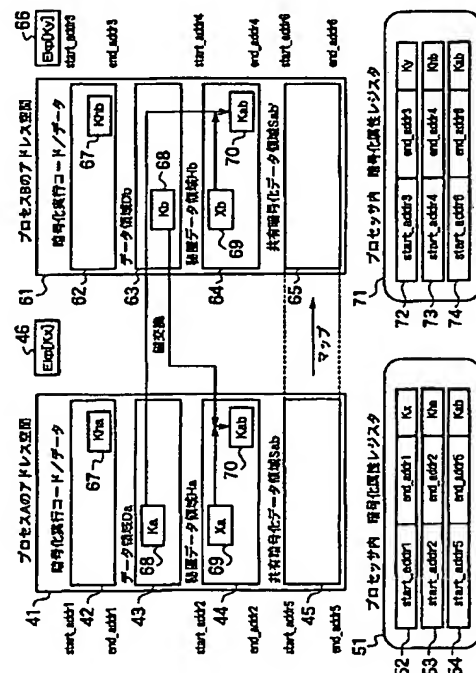
最終頁に続く

(54)【発明の名称】 耐タンパプロセッサにおける暗号化データ領域のプロセス間共有方法

(57)【要約】

【課題】 マルチタスクのプログラム実行環境を支援する耐タンパのマイクロプロセッサ上で、秘匿プログラムや秘匿データを、複数のプロセス間で安全に共有させる。

【解決手段】 プログラムおよびデータの暗号化/復号化機能を有するプロセッサの実行モードを暗号化命令実行モードに移行させ、2つのプロセスの各プロセス空間に、それぞれ独自の秘匿データ領域を生成させる。さらに、各プロセスに、鍵交換に用いるそれぞれ異なる鍵ペアを生成させ、2つのプロセス間で鍵交換を行なわせる。鍵交換に基づいて、各プロセスに共通鍵を計算させる。共通鍵と、鍵交換の過程で使用したデータを、各プロセスの秘匿データ領域に格納する。一方のプロセスに、2つのプロセス間で共有されるべき共有暗号化領域を生成させ、他方のプロセスにこの共有暗号化領域を自己のプロセス空間にマップさせる。最後に、各プロセスの共有暗号化領域と、共通鍵とを関連付けて、プロセッサ内部の暗号化属性レジスタに保存する。



【特許請求の範囲】

【請求項1】 プログラムおよびデータの暗号化および復号化機能を有するプロセッサ上で、暗号化されたデータ領域を2以上のプロセス間で共有する方法であって、前記2以上のプロセスの各々に、あらかじめ共通鍵を与えておくステップと、
プロセッサの実行モードを暗号化命令実行モードに移行させるステップと、

前記2以上のプロセスのうち、第1のプロセスに、前記共通鍵に対してのみ有効な共有暗号化領域を自己のプロセス空間に生成させるステップと、

前記2以上のプロセスのうち、その他のプロセスに、前記第1のプロセスが生成した共有暗号化領域を自己のプロセス空間にマップさせるステップと、

前記各プロセスの共有暗号化領域を、前記共通鍵と関連付けてプロセッサ内部の暗号属性レジスタに設定するステップとを含む暗号化データ領域のプロセス間共有方法。

【請求項2】 プログラムおよびデータの暗号化および復号化機能を有するプロセッサ上で、暗号化されたデータ領域を2つのプロセス間で共有する方法であって、
プロセッサの実行モードを暗号化命令実行モードに移行させるステップと、

各プロセスに、自己のプロセス空間にそれぞれ独自の秘匿データ領域を生成させるステップと、

各プロセスに、鍵交換に用いるそれぞれ異なる鍵ペアを生成させ、前記2つのプロセス間で鍵交換を行なわせるステップと、

前記鍵交換に基づいて、各プロセスに共通鍵を生成させるステップと、

前記2つのプロセス間で共有され、前記共通鍵に対してのみ有効な共有暗号化領域を生成するステップと、

前記共通鍵と、鍵交換の過程で使用されたデータとを、前記各プロセスの秘匿データ領域に保存するステップとを含む暗号化データ領域のプロセス間共有方法。

【請求項3】 前記共有暗号化領域を生成するステップは、前記2つのプロセスのうち、一方のプロセスが自己のプロセス空間に共有暗号化領域を生成し、他方のプロセスは、この共有暗号化領域を自己のプロセス空間にマップするステップを含み、

前記プロセス間共有方法は、各プロセスの共有暗号化領域とそれぞれの共通鍵とを関連付けて、プロセッサ内部の暗号属性レジスタに設定するステップをさらに含むことを特徴とする、請求項2に記載の暗号化データ領域のプロセス間共有方法。

【請求項4】 前記鍵交換のステップは、メッセージ署名を検証するステップを含み、前記共通鍵を生成するステップは、前記検証に基づき、認証された共通鍵を生成するステップを含むことを特徴とする請求項2に記載の暗号化データ領域のプロセス間共有方法。

【請求項5】 前記各プロセスが、前記共有暗号化データ領域に対して書込または読込を行なう際に、前記プロセッサ内部の暗号属性レジスタに設定された情報を参照して、外部メモリとの間でデータの暗号化および復号化を行なうステップをさらに含むことを特徴とする、請求項1～4のいずれかに記載の暗号化データ領域のプロセス間共有方法。

【請求項6】 プログラムおよびデータの暗号化および復号化機能を有するプロセッサ上で、暗号化されたデータ領域を3以上のプロセス間で共有する方法であって、
プロセッサの実行モードを暗号化命令実行モードに移行するステップと、

前記3以上のプロセスに含まれる第1のプロセスに、これらのプロセス間全体で共有される共有暗号化データ領域を生成させるステップと、

前記第1のプロセスに、前記共有暗号化データ領域に対する共通鍵を指定させるステップと、

前記第1のプロセスと、他のひとつのプロセスとの間で共有される暗号化された鍵通知領域を、他のすべてのプロセスに関してそれぞれ生成させるステップと、

前記鍵通知領域を介して、他のそれぞれのプロセスに、前記共通鍵を通知するステップと、

前記他のすべてのプロセスに、前記第1プロセスが生成した共有暗号化データ領域を自己のプロセス空間にマップさせるステップと、

前記各プロセスにマップされた共有暗号化データ領域と、通知された共通鍵とを関連付けて、前記プロセッサ内部の暗号属性レジスタに格納するステップとを含む暗号化データ領域のプロセス間共有方法。

【請求項7】 前記第1のプロセスと他のひとつのプロセスとの間で鍵通知領域を生成するステップは、

前記第1のプロセスと、他のひとつのプロセスの各々に、独自の秘匿データ領域を生成させるステップと、

前記第1のプロセスと、他のひとつのプロセスの各々に、鍵交換に用いるそれぞれ異なる鍵ペアを生成させて、鍵交換を行なわせるステップと、

前記鍵交換に基づいて、前記第1のプロセスと他のひとつのプロセスとの間で用いられる第2の共通鍵を生成させるステップと、

前記第1のプロセスと他のひとつのプロセスとの間で共有され、前記第2の共通鍵に対してのみ有効な鍵通知領域を生成するステップと、

前記第2の共通鍵と、前記鍵交換の過程でされたデータとを、前記各プロセスの秘匿データ領域に保存するステップとを含むことを特徴とする請求項6に記載の暗号化データ領域のプロセス間共有方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明はマルチタスクのプログラム実行環境を支援する機能を持つマイクロプロセッ

サに関し、特に、プログラムやデータの暗号化および復号化機能を有し、秘匿プログラムや秘匿データを複数プロセス間で安全に共有できるマイクロプロセッサに関する。

【0002】

【従来の技術】近年マイクロプロセッサの性能向上は著しく、従来の用途である計算やグラフィックにとどまらず動画、音声の再生や編集加工が可能となっている。これによりマイクロプロセッサを応用したエンドユーザ向けシステム(以下PCと呼ぶ)でユーザはさまざまな動画や音声を楽しめるようになった。PCを動画、音声の再生装置として使うことは、PCが持っている計算能力と組み合わせてゲームなどの応用ができることは当然として、固定的なハードウェアを必要としないため、すでにPCを持っているユーザにとっては安価にそれら動画、音声の再生が楽しめるという利点もある。

【0003】PCで画像や音声を扱う場合に問題となるのがその画像/音声の著作権の保護である。特に、PC上で画像データや音楽データを取り扱うソフトウェア自体の改変や不正コピーを防止すべく、耐タンパソフトウェア技術と呼ばれる手法が用いられている(David Aucsmith et.al; "Tamper Resistant Software: An Implementation", Proceeding of the 1996 Intel Software Developer's Conference)。耐タンパソフトウェア技術とは、ソフトウェアを暗号化するなどの手法により、ソフトウェアに含まれるノウハウなどを解析から守り、ソフトウェアが解読、改竄されることを防止する技術である。

【0004】しかし、耐タンパソフトウェア技術は、基本的には、プログラムのうち保護を要する部分を実行開始時には暗号化しておき、その部分を実行する前に復号化し、実行終了後に再び暗号化することにより、逆アセンブラ、デバッガなどの解析ツールによる解析を困難にする技術である。プログラムがプロセッサによって実行可能である以上、プログラムの開始時から順を追って解析していけば必ず解析することが可能である。この事実

は、PCを通じた、著作物や高度情報サービスの提供、あるいはPCへの企業、個人のノウハウを含んだプログラムの適用の妨げになっている。

【0005】さらに、ハードウェア面の問題として、オープンプラットフォームであるPCにおいて、システムのソフトウェア的土台となるはずのOS(オペレーティングシステム)自体が改変されて、攻撃の手段として用いられる可能性がある。技術を持ち、悪意のあるユーザは自分のPCのOSに改造を加え、OSの持つ特権を利用して、OSとしての正常な動作の代わりに、アプリケーションプログラムに埋め込まれている著作権保護機構を無効化したり、その機構を解析するための操作を行えるからである。

【0006】現代のOSは、CPUに備えられた実行制

御機能やメモリへの特権操作機能を利用して、複数のタスクを見かけ上は並列に処理するマルチタスク環境を作り出している。すなわち、OSはプログラムの実行に必要な資源、すなわちCPU時間とその実行に必要なメモリの割り当てを行ない、コンピュータ制御下にあるデバイス、ネットワーク、アプリケーションQoSへのアクセスを制御する。これを行なうために、OSは以下の2つの特権を有する。

【0007】第1の特権は、CPU時間の割り当てをするために、OSはアプリケーションプログラムを任意の時点で停止、再開することができる。

【0008】第2の特権は、OSは異なるアクセス速度と容量を持つ、通常は階層化されたメモリシステムをアプリケーションから隠ぺいして、フラットなメモリ空間をアプリケーションに提供するが、任意の時点でアプリケーションに割り当てたメモリ空間の内容を、異なる階層のメモリに移動することができる。

【0009】この2つの特権を持つことにより、任意の時点でOSはアプリケーションの実行状態を停止してスナップショットをとり、それを複製または書き換えて再開することが可能である。この機能はアプリケーションのもつ秘密を解析する道具にも使うことができる。

【0010】コンピュータ上のアプリケーションに対する解析を防ぐために、プログラムやデータの暗号化を行う技術はすでにいくつか知られている(Hampson, USP4, 47,902; Hartman, USP 5,224,166; Davis, USP5,806,706; Takahasi, USP5,825,878; Leonard et.al, USP 6,003,117, 特開平11-282756)。暗号化された実行コードは、秘密鍵を知っているマイクロプロセッサのみが復号化することができる。デコード(復号)状態は、マイクロプロセッサ内部に限定され、実行プロセス以外は、いかなる他プロセスあるいはOSであっても、デコードすることができない。

【0011】しかし、これらの従来の暗号化技術では、上述したOSの特権動作からプログラムの動作とデータの秘密を守るとは考慮されていなかった。したがって、従来の暗号化技術を適用したシステムでは、コンテキスト切替と呼ばれるOSの特権動作を利用することにより、実行コードの暗号化を解かなくともプログラムの解析が可能になるという欠点があった。

【0012】コンテキスト切替とは、割り込みによってプログラムの実行が中断された時や、プログラムがシステムコール呼び出しのために自発的にソフトウェア割り込み命令を呼び出した時に、OSによって行なわれる切り替え処理である。すなわち、OSは別のプログラム実行のために、その時点でのレジスタ値の集合からなるプログラムの実行状態(「コンテキスト情報」と呼ぶ)をメモリに保存する一方で、あらかじめメモリに保存されていた別のプログラムのコンテキスト情報をレジスタに復帰させる。コンテキスト切替は複数のプログラムを並行

10

20

30

40

50

動作させる上で不可欠の機能である。OSはコンテキスト切替時にレジスタの値を読むことができるので、そのプログラムの実行状態がどのように変化したかに基づいて、プログラムの行なっている動作のほとんどの部分を推定できる。

【0013】OSはまた、プログラムの実行の中断や解析ばかりでなく、メモリに移されたレジスタの情報を任意に書き換えることもできる。OSはアプリケーションの任意の状態を保存することができるため、ある時点で保存した状態から、繰り返し何度もレジスタ値を書き換えてプログラムを動作させてプログラムの動作を解析することが可能なのである。

【0014】このような問題に鑑み、出願人は、特願第2000-035898号および特願第2000-135010号において、コンテキスト切替時に、マイクロプロセッサ内部のレジスタにあるコンテキスト情報の全部または一部を自動的に暗号化して、プロセッサ外部のメモリに退避させる技術を提案した。これにより、コンテキスト切替時のOSによる攻撃を防止することができる。

【0015】一方、複数のアプリケーションやプログラム供給者が、それぞれOSとは独立して秘密を持つ技術も提案されている。米国特許第5,123,045号(Ostorovsky et.al)は、アプリケーションごとに固有の秘密鍵を持つサブプロセッサを前提とし、これらのサブプロセッサがメインメモリ上に置かれたプログラムをアクセスする時でも、そのアクセスパターンからプログラムの動作が推定され得ないシステムを開示している。このシステムは、メモリに対して操作を行う命令体系を、それとは異なる命令体系に変換し、ランダムなメモリアccessを行なわせる機構である。

【0016】しかしながら、この技術はアプリケーション毎に異なるサブプロセッサを必要とし、コスト高になるばかりでなく、このような命令体系を処理するコンパイラやプロセッサハードウェアの実装、高速化技術は、現在のプロセッサのそれとはかなり異なる。すなわち、このシステムの実現は現時点では困難であると予想されるのである。また、このようなプロセッサでは、実際に動作するコードの動作やデータを観察、追跡してもデータの内容や動作の対応関係の把握が著しく困難となる。このため、プログラムコードやデータが単純に暗号化されている先に列挙した従来技術と比較して、プログラムのデバッグが非常に困難となり、この点においても実用化に問題がある。

【0017】

【発明が解決しようとする課題】暗号化されたプログラムやその処理対象であるデータを保護する技術は、現在も研究が進められているが、基本的に、実行プロセスの解析やトレースを無効化して、他プロセスからプログラムやデータを盗み見ることを防止することに技術が注が

れている。

【0018】その一方で、正しく許可されたプロセス(プログラム)間であれば、プログラムやデータを暗号化された状態であっても、互いに共有したいという要求がある。すなわち、既存の暗号化されていないライブラリや、OSが提供するプロセス間共有メモリを用いるのと同様の感覚で、暗号化されたプログラムや秘匿データをプロセス間で共有したいという要求である。暗号化されたプログラムやデータを異なるプロセス間で共有するためには、正しいプロセス間で鍵情報をいかにして安全に共有するかが問題となる。

【0019】暗号化されたプログラムやデータの復号状態をプロセッサ内部に限定することのできるプロセッサにおいて、あるプロセスが作成する暗号化されたメモリ領域を他プロセスと共有したい場合、OSの提供するメモリ共有機構を利用するだけでは、意味のあるデータ共有はできない。暗号化が施されたメモリ領域の共有を要求する他プロセスは、この領域のための必要な鍵を知らないからである。

【0020】暗号化されたメモリ領域の読み書きは、このメモリ領域に対応する鍵を使用することによって行なわれる。鍵は、プロセッサ内部で秘密裏に保持されている。そこで、他プロセスにおいても同一の鍵をプロセッサ内部に共有することができれば、意味のある読み書きが可能となる。ただし、この鍵共有が、2プロセス間で安全に行なわれなければ意味がない。

【0021】鍵を共有するには、互いに予め鍵を知っているプロセス間であれば、鍵をプログラム内部に暗号化した状態で埋め込んでおくことにより安全に共有することが出来る。しかしながら、鍵を予め埋め込むことができない、あるいは、事前に通知したくないプロセス(プログラム)間で鍵を共有したい場合は、プロセス間で安全な鍵交換シーケンスを確立する必要がある。

【0022】これまでに、ネットワークを経由して安全な鍵交換シーケンスの確立を目指した技術は、公開鍵暗号方式の登場により数多く登場してきている。ネットワーク上で交換される鍵情報は、それ自体盗み見ることはできるが、各計算機内部で保持される秘密鍵を知らなければ攻撃は難しい。ネットワーク経由の鍵交換の場合、計算機内部での鍵交換/鍵算出に使用される一時データが平文で扱われていても、これらがネットワークに流出しない、あるいはネットワークから盗み出せないことを前提に実現されていた。

【0023】本発明においては、ネットワークの鍵交換シーケンスとは別に、計算機内部でプロセス間の鍵交換を行なう必要があり、その鍵交換や鍵算出時に使用される一次データも計算機内部に置かれ得る場合を前提とする。

【0024】すなわち、本発明の第1の目的は、実行プロセスの秘密を解析や意図的な改変から守ることのでき

10

20

30

40

50

るプロセッサ（耐タンバプロセッサ）において、2つのプロセス間で、暗号化されたデータ領域を安全に共有する方法を提供することにある。

【0025】本発明の第2の目的は、3以上のプロセス間で、安全な通信路を利用して、暗号化されたデータ領域を共有する方法を実現することにある。

【0026】

【課題を解決するための手段】上記目的を達成するために、本発明では、暗号化プログラムの実行コードやそこで処理される暗号化データを正当な実行プロセスだけが利用できる実行環境の耐タンバのマイクロプロセッサを前提とする。

【0027】第1の目的、すなわち耐タンバのマイクロプロセッサ上で暗号化されたデータ領域を2つのプロセス間で共有させる方法を実現するためには、あらかじめ2つのプロセスに共通鍵を与えておく方法と、共有されるべき暗号化データ領域を作成する際に共通鍵を生成し、共有相手のプロセスに生成した共通鍵を通知する方法とがある。

【0028】前者の方法は、プロセス間共有を行なう各プロセスにあらかじめ共通鍵を与えておくので、任意の数のプロセス間での暗号化データ領域の共有に拡張することができる。この方法は、プログラムおよびデータの暗号化／復号化機能を有するプロセッサ上で暗号化されたデータ領域の共有を行なう2以上のプロセスの各々に、あらかじめ共通鍵を与えておくステップと、プロセッサの実行モードを暗号化命令実行モードに移行させるステップを含む。そして、2以上のプロセス中の第1のプロセスに、与えられた共通鍵に対してのみ有効な共有暗号化領域を自己のプロセス空間に生成させ、その他のプロセスに、第1のプロセスが生成した共有暗号化領域を自己のプロセス空間にマップさせる。さらに、各プロセスの共有暗号化領域を、それぞれに与えられた共通鍵と関連付けてプロセッサ内部の暗号属性レジスタに設定する。

【0029】後者の方法は、2つのプロセス間で共有する暗号化データ領域用の鍵情報を、各プロセスが事前に共有できない場合に、暗号化データ領域の作成元であるプロセスがその都度共通鍵を生成し、他方のプロセスにこの共通鍵を通知するものである。この方法は、まず、プログラムおよびデータの暗号化／復号化機能を有するプロセッサ上で、プロセッサの実行モードを暗号化命令実行モードに移行させる。次に、2つのプロセスの各プロセス空間に、それぞれ独自の秘匿データ領域を生成させる。各秘匿データ領域のアドレスを、それぞれの対応する秘匿データ領域用の鍵と関連付けて、プロセッサ内部の暗号化属性レジスタに設定する。さらに、各プロセスに、鍵交換に用いるそれぞれ異なる鍵ペアを生成させ、2つのプロセス間で鍵交換を行なわせる。鍵交換に基づいて、各プロセスに共通鍵を生成させる。そして、

2つのプロセス間で共有され、前記共通鍵に対してのみ有効な共有暗号化領域を生成する。最後に、共通鍵と、鍵交換の過程で使用されたデータとを、各プロセスの秘匿データ領域に保存する。

【0030】共有暗号化領域を生成するには、2つのプロセスのうち、一方のプロセス（オーナープロセス）が自己のプロセス空間に共有暗号化領域を生成し、共有を要求する他方のプロセス（クライアントプロセス）は、この共有暗号化領域を自己のプロセス空間にマップする。この場合、各プロセスは、共有暗号化領域と、各々の共通鍵とを関連付けて、プロセッサ内部の暗号化属性レジスタに設定する。

【0031】この方法では、共通鍵は秘密の状態で生成され、それに使用されたデータも共通鍵自体も秘匿データ領域に保存されるので、共通鍵を知っている2つのプロセス以外のプロセス、あるいはOSからですら、アクセスすることができない。

【0032】暗号化データ領域の共有の信頼性をさらに高めたい場合には、2つのプロセス間での鍵交換を、メッセージ署名の検証に基づいて行なう。具体的には、各プロセスは、乱数を発生させ、この乱数と、公共機関が作成した証明書とを、互いに相手のプロセスに送って認証を要求する。認証が確認されたなら、各プロセスは鍵交換用のフェーズ値を計算し、このフェーズ値に署名を付けて、相手プロセスに送る。各プロセスは、送られてきたフェーズ値と、自己が発生した認証済みの乱数とを用いて、共通鍵を生成する。

【0033】この方法によれば、たとえ2つのプロセス間に悪意のプログラムが介在する場合でも、そこからの攻撃を防止することができる。

【0034】第2の目的、すなわちプログラムやデータの暗号化／復号化機能を有するプロセッサ上で、3以上のプロセス間で暗号化データ領域の共有する方法を実現するために、まず、プロセッサの実行モードを暗号化命令実行モードに移行させる。次に、3以上のプロセスの中の第1のプロセスに、これらのプロセス間全体で共有される共有暗号化データ領域を生成させる。さらに、前記第1のプロセスに、生成した共有暗号化データ領域に対する共通鍵を指定させる。次に、この第1のプロセスと、他の任意のひとつのプロセスとの間でのみ共有される暗号化された鍵通知領域を、他のすべてのプロセスに関してそれぞれ生成させる。共通鍵は、2プロセス間で共有される暗号化された鍵通知領域を介して、他のプロセスに通知される。そして、他のすべてのプロセスに、第1プロセスが生成した共有暗号化データ領域を自己のプロセス空間にマップさせる。最後に、各プロセスにマップされた共有暗号化データ領域と、通知された共通鍵とを関連付けて、プロセッサ内部の対応する暗号化属性レジスタに格納する。

【0035】第1のプロセスと、他の任意のプロセスと

の間で共有される暗号化された鍵通知領域の生成は、第1の目的と関連して上述した2プロセス間の共有方法を利用して生成することができる。

【0036】このように、2プロセス間で共有される鍵通知領域は、これらのプロセス間の安全な通信路として機能し、全体で最終的に共有される共有暗号化データ領域用の共通鍵を、安全な通信路を介して各プロセスに通知することができるので、3以上のプロセス間においても、ひとつの暗号化データ領域を安全に共有することができる。

【0037】

【発明の実施の形態】本発明の実施の形態として、2プロセス間での暗号化データ領域の共有と、3以上のプロセス間での暗号化データ領域の共有について述べる前に、まず、本発明が前提とするマイクロプロセッサのアーキテクチャを説明する。本発明が適用されるマイクロプロセッサは、平文と暗号化の両方の実行プログラムおよびデータを扱うことのできるアーキテクチャを有し、かつ暗号化プログラムの実行コードや、処理対象である暗号化データを正しい実行プロセスのみが利用できる耐タンバプロセッサであることを前提とする。安全性の面から言えば、平文プログラムの実行を許可しない専用マイクロプロセッサを挙げることができるが、そうすると可搬性が低く、暗号化を施す前のプログラムの作成・デバッグや、非暗号化プログラムを導入する可能性が排除されてしまうので好ましくない。

【0038】図1は、本発明が適用されるマイクロプロセッサの基本構成図である。マイクロプロセッサ1は、プロセッサコア3と、命令TLB2と、例外処理部8と、データTLB9と、2次キャッシュ10とを含む。プロセッサコア3は、命令実行部4と、1次キャッシュ5と、コード・データ暗号化／復号化部6と、バスインタフェースユニット7を含む。コード・データ暗号化／復号化部6は、暗号化属性レジスタ20を有する。

【0039】通常のプロセッサと大きく異なる点は、プロセッサコア3の内部にコード・データ暗号化／復号化部6を搭載していることである。暗号化された実行コードやデータは、バス上では暗号化された状態で流れる。プロセッサコア3に入力された実行コードやデータは、コード・データ暗号化／復号化部6によって、1次キャッシュ5へフェッチされる前に復号化される。プロセッサコア3から再度バス上へ流れ出るときには、暗号化されて出力される。

【0040】命令実行部4は、図示はしないがレジスタ集合を有し、レジスタ内部では、平文データを処理する。マルチタスクをサポートするOSの場合、偽造OSなどによって強制的にコンテキスト切替えを生じさせてレジスタ情報を盗み見ようとする攻撃が考えられる。しかし、出願人による特願第2000-035898号や特願第2000-135010号に開示された方式を採用することによって、コン

テキストの保存復帰時に自動的に暗号／復号化が働き、防御することができる。

【0041】コード・データ暗号化／復号化部6の内部には、公開鍵暗号方式に基づいたプロセッサ固有の秘密鍵が保持されており、いかなる特権のプログラムであってもこれを読み出すことはできない。この秘密鍵は、対応する公開鍵で暗号化されたプログラムやデータを復号する際に、自動的にプロセッサ内部で使用される。

【0042】図1のマイクロプロセッサアーキテクチャを採用することによって、悪意のあるユーザやプログラムによる解析や改変からプログラムを保護することができる。暗号化されたプログラムのコードやデータは、特定の鍵を知っている正しいプロセッサでなければ復号し、利用できないからである。

【0043】保護したいコードやデータは外部記憶装置やメモリ上に存在するときには暗号化されており、プロセッサ内部の一次キャッシュ内およびレジスタに読み込まれている間のみ復号化される。再びキャッシュからメモリ上へ書き戻される必要がある場合には暗号化処理がプロセッサ内部で自動的に施されるのは、上述したとおりである。このようにして、暗号化された命令コードやデータが復号（平文）化される状態をプロセッサ内部に閉じ込め、レジスタ上の平文データがコンテキスト切り替え時に露呈するのを防ぐ。

【0044】図2は、暗号化属性レジスタ20の構成例を示す。暗号化属性レジスタ20は、復号化（暗号化）されるべき各プログラムやデータに対応する鍵を、プロセッサ内部で個別に保持する複数の秘匿レジスタである。いかなる特権を持ったプログラムからもこれを読み出すことはできず、正しいプログラムの実行期間中のみ、マイクロプロセッサ1が利用できる。

【0045】暗号化属性レジスタ20は、暗号化された領域の開始アドレス、終端アドレス（もしくは領域サイズ）、適用暗号アルゴリズム、鍵格納データ30へのオフセット、鍵サイズなどの情報が登録される。鍵格納データ30には、鍵の値や署名が置かれる。

【0046】マイクロプロセッサ1が暗号化された実行コードを実行する際には、暗号化命令実行モードと呼ばれる実行モードへ移行する。暗号化命令実行モードでは、マイクロプロセッサ1が、対応する暗号化属性レジスタ20に設定された復号鍵を利用して、暗号化実行コードを自動的に復号化し、命令実行を処理する。この際、スタックに積まれるデータや他のデータ領域への書き込みも自動的に暗号化され、読み込み時に自動的に復号化される。復号化され平文化された状態が存在するのは、命令実行部4のレジスタや1次キャッシュライン上のみであり、データがバスを介して外部メモリへ出力される際には、再び暗号化状態になる。

【0047】暗号化データへのアクセスは、通常、暗号化命令実行モードにて行われるが、平文命令実行モード

(通常の命令実行モード)から、暗号化されたデータへのアクセスを行い、外部メモリ上では暗号化されたままデータを読み書きを自動的に行うことも可能である。しかしながら、一般に、暗号化されるべきデータにアクセスする際には、プロセッサの命令実行シーケンスがトレースされることを防ぎ、解析されにくくする目的で、暗号化命令実行モードへ移行して実行することが望まれる。

【0048】プロセス切替えを行う際には、実行プロセスのコンテキスト(レジスタ情報など)が外部メモリへ保存されるが、そのうち、他プロセスから秘匿されるべき暗号化属性レジスタ情報などは、マイクロプロセッサ1が暗号化処理を行った上でメモリ上に保存される。

【0049】あるいは、暗号化属性レジスタ20は実行対象プロセスのIDを伴い、プロセス切り替え後も暗号化属性レジスタ20の情報をマイクロプロセッサ1の内部に保持し続ける構成としてもよい。この場合、このプロセスの実行時以外(すなわち他プロセス実行時には、プロセッサが利用できないことを保証した形式でコンテキスト切替え処理を行う。

【0050】上述した特徴を持つマイクロプロセッサを、本発明では耐タンバプロセッサと呼称する。

【0051】次に、本発明が前提とする耐タンバプロセッサ上でのプログラム実行動作について説明する。

【0052】まず、マイクロプロセッサ(耐タンバプロセッサ)1を内部に有する計算機上で、保護されたプログラムを実行する手順を述べる。プログラム実行に先立って、該計算機内に保護対象プログラムがインストール済みであるとする。また、計算機上で、プロセッサ1が暗号化された実行コードを復号するために必要な鍵は、プログラムの作成元あるいは発行元などによって、プロセッサ固有の秘密鍵に対応する公開鍵を用いて暗号化され、インターネット経由あるいは配付メディアに添付するなどの手段により、事前に計算機にダウンロード、インストールされているものとする。あるいは、プログラマ

setk セグメントID 公開鍵で暗号化された鍵 … (1)

暗号化されている領域の情報をセグメントIDという領域のIDを用いて指定している。

【0058】セグメントIDとは、マイクロプロセッサ1が領域を識別するためのIDであり、領域の先頭アドレス、あるいは先頭アドレスと領域サイズ(もしくは終端アドレス)の組み合わせを格納したセグメント情報への識別子に相当する。

【0059】一般に、プログラムは、ライブラリを含む様々な実行コード(オブジェクト)がリンクされることによって構成される。実行コード(オブジェクト)には、命令部、データ部、リンク情報部(デバッグ時にはデバッグ情報も含みうる)が含まれ、これらに対しても、同一の鍵による暗号化もしくは別々の鍵による暗号化を施すことができる。

*ム作成元や発行元により、ユーザ所有のプロセッサに対応する公開鍵を用いて、プログラム実行に必要な復号鍵を保護対象プログラム内部にあらかじめ暗号化して埋め込む場合もある。

【0053】マイクロプロセッサ1は、暗号化された実行コードを復号する際には、この鍵を利用する。このため、保護されたプログラムを実行する初期段階で、かつ、暗号化実行コード部に突入する以前に(平文実行コード内で)、この実行コードの復号化鍵を、プロセッサ内部の暗号化属性レジスタ20に平文状態で設定しておく必要がある。復号化鍵は、プロセッサ固有の公開鍵で暗号化されているが、下記の命令を発行することによって、プロセッサ内部でプロセッサの秘密鍵によって解読され、暗号化属性レジスタ20に平文状態で安全に設定することができる。

【0054】setck プロセッサの公開鍵で暗号化された復号鍵データ上記の例では、実行中のプロセスに対して1つの復号鍵のみを設定可能なシステムを対象にした命令例を示している。すなわち、このシステムの場合は、プログラム内部に平文実行コード、暗号化実行コードが複数混在する場合であっても、同一の復号鍵を共有することになる。

【0055】プログラムによっては、複数の作成元によって別々に作成された複数種類の実行コード(オブジェクト)を組み合わせて構築されている場合がある。近年ではこうした形態のプログラミングを行うのが一般的になっており、この場合、種々の実行コードが別々の暗号化鍵によって暗号化されている状態に対応したシステムを用意しなければならない。すなわち、各実行コードの領域毎に、別々の暗号化処理を施したプログラムを作成し、それを復号化して実行するために必要な復号鍵も、個別に用意しておく必要がある。

【0056】この場合、暗号化属性レジスタ20への実行コード復号鍵の設定命令は、以下のようになる。

【0057】

setck セグメントID 公開鍵で暗号化された鍵 … (1)

【0060】プログラムの実行時には、スタック領域や動的に割り当てられるデータ領域が、OSやプログラムローダ、ダイナミックリンカからの支援によって設定され、これによりプログラムが実行可能状態となる。

【0061】プログラムの実行を開始するには、まず、プログラムローダ(およびOS)によりプログラムをメモリ上へロード(マップ)し、実行コードやスタック、データなどの各領域を初期設定する。ダイナミックライブラリを使用する際には、ダイナミックリンカやダイナミックライブラリが実行時に適切に呼び出されるよう前処理をしておく。

【0062】実行コードやスタック、データなどの各領域に対して暗号/復号化を指定する場合には、上述したように、領域ごとに対応する鍵を指定し、マイクロプロ

セッサ1内の暗号化属性レジスタ20にそれぞれ平文状態で設定しておく。

【0063】領域の暗号関連情報（各領域のサイズ、マップ先、属性など）は、プログラム作成元や発行元によってプログラム内部にあらかじめ埋め込まれてもよいし、あるいは、プログラムとは別に計算機にダウンロードされてもよい。プログラム内部に暗号関連情報が埋め込まれる場合は、どの位置に格納されるかは、実行バイナリ形式に依存する。

【0064】各領域に対応する鍵は、命令(1)を用いて暗号化属性レジスタ20に設定されるが、この設定は、プログラムローダー（およびOS）によって、プログラムのロード、アドレス空間へのマップ時に行われる。

【0065】動的に暗号化対象となるデータ領域を新規に作成する場合には、その時点で命令(1)を使用して暗号化属性レジスタ20への設定を行う。この動作に関しては、後述する暗号化データ領域のプロセス間共有にて詳述する。

【0066】図3は、複数種類の実行コードを組み合わせたプログラムを実行する際の、暗号化属性レジスタ20への鍵設定の例を示す。図3において、プロセッサ固有の公開鍵 K_p によってそれぞれ暗号化された実行コード（オブジェクト）の鍵 $11 \sim 14$ （ $E_{Kp}[Kx]$ ）がエントリされ、これらの鍵 K_x で暗号化されているプログラムの各実行コード領域がプロセス空間31に配置されている。暗号化属性レジスタ20は、対応する領域の開始アドレス、終端アドレス、鍵を平文状態で格納する。暗号化属性レジスタ20の構成は、説明の便宜上、簡略化してある。

【0067】図3の例では、プログラムは、鍵 K_a で暗号化されている実行テキスト領域 T_a とデータ領域 D_a 、鍵 K_b で暗号化されている実行テキスト領域 T_b 、鍵 K_c で暗号化されている実行データ領域 D_c 、暗号化命令実行モード下で、鍵 K_d で暗号化されてデータを格納する（暗号化）スタック領域 S から構成されている。ここでは、プロセッサの公開鍵で暗号化された鍵エントリ $11 \sim 14$ は、プログラムの外部から供給される。命令(1)を使用することによって、マイクロプロセッサ1の内部で、各鍵エントリ $11 \sim 14$ から、鍵 K_a 、 K_b 、 K_c 、 K_d をそれぞれ復号し、プロセス空間に配置された各暗号化領域に対して、暗号化属性レジスタ20の各エントリ $22 \sim 26$ に領域情報、鍵情報などを登録する。

【0068】次に、平文の命令実行モードから暗号化命令実行モードへの移行処理について説明する。この移行処理の方法には、明示的移行処理命令記述方法と、自動移行処理方法の2通りがある。

【0069】(1) 明示的移行処理命令記述方法

まず、プログラムの先頭から命令を順次実行していくと、保護すべき対象となる実行コードの直前で、プロセ

ッサの実行モードを暗号化命令実行モードへ移行させる命令が明示的にコード内に記述されている場合がある。マイクロプロセッサ1がこの移行命令を実行することによって、以降の実行コードに対しては、暗号化された実行コードと見做して、実行コードを復号しながらデコード、実行する。

【0070】暗号化命令実行モードへの移行命令の例は、以下ようになる。

【0071】

10 モード移行命令：cryptomode（飛び先アドレス）
同様に、暗号化命令実行モードから通常の平分命令実行モードへ戻す命令は、以下ようになる。

【0072】

モード解除命令：retmode（戻り先アドレス）

飛び先アドレス、および、戻り先アドレスは省略可能で、それぞれ、省略時には、直後の命令の直前でモード移行処理が行なわれる。

20 【0073】暗号化モードへの移行命令が発行されてから、モード解除命令が発行されるまでの期間、マイクロプロセッサ1は、プログラムカウンタで指し示される実行コードを暗号化されたものとみなして復号、デコード、実行を続ける。

30 【0074】この方式では、一度暗号化命令実行モードに突入すると、途中で、他の暗号鍵で暗号化された実行コード内の関数ヘジャンプ/コールする場合には、呼び出し先でも同様な暗号化命令実行コードへの移行命令を記述しておく必要がある。さらに、呼び出し先から戻る場合にも、モード解除命令を記述する必要があるが、このままでは、呼び出し元の実行モードへの移行が自動的に行えない。そこで、ある暗号化命令実行モードに入ってから別の暗号化命令実行モード（あるいは平文命令実行モードでもよい）に移行する際（コール時）には、スタックに、戻りアドレスのほかに呼び出し元の暗号化属性レジスタ20のIDを追加しておく。あるいは、バックリンク情報としてプロセッサ内部で保持し、フラグを設定しておき、リターン時にこの情報を利用して元の命令実行モードへ回復させる。

40 【0075】この方式では、プログラム開発ツール（コンパイラ、リンカ、暗号化ツールを含む）などを用いて、実行プログラム内の暗号化領域として指定した領域の直前/直後に、上記の命令実行モード移行命令を自動的に挿入し、モードの移行が破綻しないよう正しく管理することが必要となる。明示的なモード移行命令を任意の位置に挿入することによって、暗号化される領域の粒度を比較的細かく指定することができるが、暗号化命令実行モードへの移行位置をプログラム内に明記しているため、ハッカーなどから攻撃対象となるコード位置を教えているのに等しい。本当に暗号化すべき部分のみモード移行を行い、それ以外は極力平文実行モードで高速なプログラム実行を期待する場合には有効かもしれない

が、解析に対する強度は下がってしまうという欠点がある。なお、この場合、予め暗号化領域に対する暗号化属性レジスタを設定しておく必要はない。

【0076】(2) 領域アクセス時に暗号化属性に応じた自動移行処理方法

一方、上述の明示的モード移行命令を使用せず、あらかじめ暗号化領域をメモリ上にマップする際に、暗号化属性レジスタを設定しておき、暗号化対象コード領域での実行や暗号化対象データ、スタック領域への読み書き時に自動的に実行モードへの移行を判別、必要な場合には実行モードの切り替えを行う方式もある。

【0077】この場合には、命令(1)を用いて、予め実行コードの暗号化された領域情報をプロセッサ内部の暗号化属性レジスタ20に記憶させておくことになる。

【0078】明示的なモード移行命令やモード解除命令はなく、これらの指定領域に突入したり、指定領域の外の実行コード部へ実行が移行した際には、自動的にモード切替えが行われる。実行中の暗号化された実行コードから移行しようとする先の実行コードが別の鍵で暗号化された領域である場合には、移行先の暗号化実行コードに対応したモード移行が自動的に実行される。暗号化対象のデータ、スタック領域に対しても同様な判別、実行が行われる。

【0079】以上の構成を有する耐タンバプロセッサ(マイクロプロセッサ1)を前提として、本発明のプロセス共有方式を説明する。

【0080】<第1実施形態>第1実施形態においては、2つのプロセス間でのみ使用可能なデータ領域、すなわち暗号化データ領域の共有方式について説明する。

【0081】ここで生成される共有データ領域に対して読み書きされるデータは、マイクロプロセッサ内部のハードウェアで暗号化/復号化され、鍵を知らない他プロセスからは秘匿される。

【0082】異なるプロセス間で、ある暗号化されたデータ領域を共有するためには、通常のOSがサポートするメモリ共有手段のほかに、そのデータ領域に対して使用される暗号/復号化鍵を2つのプロセス間で共有させる手段が必要がある。

【0083】これを実現するのに2つの方法がある。

【0084】第1の方法は、最も単純な方法として、共有すべき暗号化データ領域用の鍵を、プロセス間共有するプログラム間で事前に知っている、あるいは、共有できる状態にしておくことである。

【0085】第2の方法として、鍵を事前に共有できない場合、暗号化データの共有を要求する正しいプロセスに対して、鍵情報を安全に通知する方法である。

【0086】第1の方法、つまり、事前の鍵の共有については、同一プログラムに関してであれば、複数のプロセス間で暗号化データを共有するために、暗号化データ領域用の暗号/復号鍵を、あらかじめ同一のものに設定

することが可能である。鍵の設定方法には、(i)実行コード内の暗号化されたデータ領域内に鍵を直接格納しておく、あるいは、(ii)必ず同一の鍵を生成できるような初期値、アルゴリズムを利用して鍵を一意に決定しておくことによって、鍵を暗号化データ共有領域に対応する暗号化属性レジスタに設定する方法がある。

【0087】暗号化属性レジスタへの鍵の設定は、前述した命令(1)でできる。命令(1)では、暗号化属性レジスタに指定する鍵は、プロセッサ固有の公開鍵で暗号化されていたが、プロセス間の共通鍵を設定する場合、すでに命令実行コードおよび鍵自体が暗号化されているため、以下の命令によって、直接鍵を設定することも可能である。

【0088】setik セグメントID 鍵 … (2)

後は、通常の共有メモリと同様に、OSが提供するメモリ共有手段を利用して、暗号化データ領域の割り当て、他プロセスからのアタッチを行えばよい。メモリ共有を要求する他プロセスは、対象となる暗号化データ領域に対応する鍵をすでに知っているため、領域へのアクセス権さえ獲得できれば、暗号化データ領域の作成元プロセス(オーナープロセス)と同様のメモリアクセスが、この暗号化データ領域に対して可能となる。

【0089】第2の方法、すなわち、プログラム作成時に暗号化データ領域用の鍵情報を事前に共有できない場合には、暗号化データ領域を作成したプロセス(「オーナープロセス」と称する)が、暗号化/復号化に使用する鍵情報を決定し、データ共有を要求する正当な他プロセス(「クライアントプロセス」と称する)に対して、鍵情報を安全に通知する必要がある。

【0090】この場合、まず、暗号化データ領域用の鍵(の初期値)を、領域を生成するオーナープロセスが決定する必要がある。

【0091】鍵の初期値決定には、第1の方法と同様に、実行コード内の暗号化されたデータ領域内に直接格納しておいた鍵の初期値を利用するか、あるいは、必ず同一の鍵を生成できるような初期値、およびアルゴリズムを用いて、鍵を一意に決定しておくことによって、毎回同一の鍵の初期値を使用することが可能であるが、これ以外にも、乱数発生ルーチン(ソフトウェア)や乱数発生器(ハードウェア)を呼び出すことによって初期値を設定し、動的に鍵の初期値を生成してもよい。この場合、プログラム実行ごとに毎回異なる鍵を生成/使用することにより、解析からの強度を高めることになる。この、鍵の初期値決定シーケンスは、暗号化命令実行モードにて行われることが望ましい。

【0092】オーナープロセスが鍵の初期値を決定したならば、暗号化データ領域の共有を要求している他プロセスと鍵交換を行うことによって、2プロセス間でのみ利用可能な暗号化データ領域用の鍵を算出する。

【0093】図4および5は、共通鍵をその都度生成す

ることによって暗号化データ領域を共有する場合の、処理手順を示す図である。図4および5に示す例では、Diffie-Hellman (D-H) 鍵交換シーケンスを応用する。D-H法では、 $K_{ab} = (\alpha \wedge X_a) \wedge X_b = K_a \wedge X_b = K_b \wedge X_a$ が、 $f(K_a, K_b)$ からは計算できないという特徴を利用して、2者間でのみ利用可能な鍵を生成することが可能である。ここで、 K_{ab} は共通鍵を表わし、 K_a 、 K_b の値と、鍵交換のための鍵 X_a 、 X_b をべき指数とするべき乗計算で算出される。

【0094】図4および5において、オーナープロセスAとクライアントプロセスBが実行中である。オーナープロセスAは、プロセス間で共有する共有暗号化データ領域 S_{ab} を生成するプロセスであり、クライアントプロセスBは、暗号化データ領域 S_{ab} の共有を要求するプロセスである。

【0095】まず、オーナープロセスAは、暗号化実行命令モードに移行すべく、実行プロセス空間（あるいはアドレス空間）に暗号化実行コード／データ領域42を生成し、この領域42に対して、たとえば外部メモリに暗号化されて格納されている鍵 $E_{kp}[K_x]$ を指定する（ステップS81）。鍵 K_x は、マイクロプロセッサ1に固有の秘密鍵 K_p で復号化され、復号化された鍵 K_x と、暗号化実行コード／データ領域42の開始アドレスおよび終端アドレスとを関連付けて、暗号化属性レジスタ52に設定する。

【0096】オーナープロセスAはさらに、秘匿されるべき暗号化データのための秘匿データ領域 H_a を、自己のプロセス空間に独自に割り当て、この秘匿データ領域 H_a に対して、秘匿データ領域用の鍵 K_{ha} を指定する（ステップS82）。秘匿データ領域用の鍵 K_{ha} は、暗号化されたプログラム内にあらかじめ埋め込まれている場合は、これを用いるが、乱数発生器で初期値を生成し、鍵 K_{ha} を動的に作成してもよい。鍵 K_{ha} が動的に作成された場合は、オーナープロセスAの暗号化実行コード／データ領域42に保存される。秘匿データ領域 H_a の開始アドレス、終端アドレスは、この領域用の鍵 K_{ha} と関連付けて、暗号化属性レジスタ53に設定される。

【0097】次に、オーナープロセスAは、D-Hにより、鍵交換に必要な鍵 K_a および X_a を算出する（ステップS83）。具体的には、まず、プロセスAの鍵 K_a を計算するために必要な情報（ α と q ）を生成する。これらの値の生成手段は、乱数発生による等、任意の方法をあらかじめ決めておく。さらに、鍵交換に必要なもうひとつの鍵 X_a を決定し、これを秘匿メモリ領域 H_a 44に保存する。これらを用いて、オーナープロセスAの鍵 $K_a (= \alpha \wedge X_a \bmod q)$ を算出し、鍵 K_a を暗号化されない通常のデータ領域 D_a 43に格納する。この計算途中で使用されるデータはすべて、暗号化されたスタック領域（不図示）、または秘匿データ領域 H_a 上に置かれる。

【0098】プロセスAの鍵 K_a は公開鍵のような扱い

で、プロセス空間を超えて、バスやメモリ場を平文の形で交換される。一方、鍵交換のための鍵 X_a は、秘密鍵のような扱いで、暗号化された秘匿領域 H_a に置かれている。

【0099】上記のステップは、クライアントプロセスBにおいても行なわれる。すなわち、プロセスBは、対応するプロセス空間に暗号化実行コード／データ領域62を指定し（ステップS91）、さらに独自に秘匿データ領域 H_b 64を生成して、秘匿データ領域 H_b 64用の鍵 K_{hb} を指定する（ステップS92）。これらの領域のアドレスおよび対応する鍵は、暗号化属性レジスタ71の対応するエントリ72、73に設定される。さらに、D-H法により、プロセスAとの鍵交換に必要な鍵 K_b と X_b を算出する（ステップS93）。鍵 K_b は平文状態で通常のデータ領域 D_b 63に置かれ、鍵 X_b は、秘匿データ領域 H_b 64に置かれる。

【0100】次に、オーナープロセスAは、鍵 K_a をプロセスBに送信する（ステップS84）。鍵 K_a は公開鍵のような扱いなので、ライブラリやOSが提供するAPIなどの、任意のプロセス間通信機能を利用して送信することができる。一方、クライアントプロセスBも、同様のプロセス間通信機能を利用して、鍵 K_b をプロセスAに送信する（ステップS94）。

【0101】オーナープロセスAは、プロセスBから受信した鍵 K_b と、自己の持つ鍵 X_a とを用いて、共通鍵 $K_{ab} (= K_b \wedge X_a)$ をべき乗計算し、この共通鍵を秘匿メモリ領域 H_a に保存する（ステップS85）。

【0102】同様に、クライアントプロセスBは、プロセスAから受信した鍵 K_a と自己が有する鍵 X_b とを用いて、共通鍵 $K_{ab} (= K_a \wedge X_b)$ を算出し、これを秘匿メモリ領域 H_b に保存する（ステップS95）。

【0103】次に、オーナープロセスAは、プロセスAとBとからのみアクセスが許可される共有暗号化データ領域 S_{ab} を作成する（ステップS86）。共有領域 S_{ab} は、ライブラリやOSが提供するメモリ割り当て手段を利用して作成される。共有領域 S_{ab} の開始アドレス、終端アドレスは、共通鍵 K_{ab} と関連付けて、暗号属性レジスタ54に設定される。クライアントプロセスBは、オーナープロセスAが作成した共有暗号化データ領域 S_{ab} を、OSが提供するメモリ共有手段を利用して、自身のプロセス空間にマップする（ステップS96）。このメモリ共有情報のやりとりは、暗号化されている必要はない。プロセスBの空間にマップされた共有暗号化データ領域 $S_{ab'}$ に関しても、開始アドレス、終端アドレスを、プロセスBが有する共通鍵 K_{ab} と関連付けて、暗号化属性レジスタ74に設定する。

【0104】このような状態にした上で、プロセスA、Bともに、共有暗号化データ領域 S_{ab} （または $S_{ab'}$ ）への読み書きを自由に行なうことができる（ステップS100）。共有暗号化データ領域へのアクセスは、マイ

10

20

30

40

50

クロプロセッサ(耐タンバプロセッサ)1の暗号化属性レジスタを介して行なわれるため、共通鍵を知っているプロセスAおよびBのみがアクセス可能である。これ以外の共通鍵を知らない他プロセスからはアクセスできず、2プロセス間だけで暗号化データ領域の共有が安全に行われる。

【0105】また、各プロセスが、共有暗号化データ領域に対してデータの書き込み/読み込みを行なう際には、プロセッサコア内部の暗号化属性レジスタの情報を参照して、1次キャッシュと外部メモリとの間で、自動的にデータの暗号化/複号化を行なう。これにより、暗号化プログラムにおける処理データの秘密も保護される。

【0106】<第2実施形態>第1実施形態では、2つのプロセス間で鍵交換を行い、共通鍵をそれぞれの暗号化属性レジスタに登録することによって、共有暗号化データ領域が実現された。第2実施形態では、鍵交換に認証を行なうことにより、さらに安全な暗号化データ領域の共有方法を提供する。

【0107】第1実施形態の方式では、悪意のある改造OSなどのプログラムが鍵交換を行なう2つのプロセス間に介在した場合、man in the middle attackと呼ばれる攻撃がなされる可能性がある。すなわち、介在プログラムが、鍵の生成に必要な情報を偽り、鍵を任意に操って暗号化データを自由に盗み見る、あるいは改ざんするおそれがある。

【0108】悪意のある改造プログラムがシステムに挿入される余地のないシステムであれば、第1実施形態の鍵交換方式を適用しても問題はないが、PC上の汎用OSのようにプログラムを自由に追加できるようなシステムの場合は、攻撃を受ける可能性がある。

【0109】このような攻撃から鍵交換シーケンスを防御するためには、プログラムが使用する公開鍵などの情報を公共機関などを通じて認証/発効した証明書を用意し、これを用いた正当な公開鍵でなければ正しく交換できないことを保証する方式を利用すればよい。

【0110】以下に、公開鍵暗号技術である電子署名アルゴリズムと、D-H鍵交換アルゴリズムを使用して防御する方法を、図6を参照して説明する。鍵交換シーケンス中に用いられる暗号化アルゴリズムには、任意のものを使用できる。

【0111】この方法を実行中、実行プロセスは暗号化命令実行モードに移行し、鍵交換シーケンスの間に作成/使用されるデータは、各プロセス内で用意される秘匿メモリ領域Ha、Hbに置かれるものとする。

【0112】図6において、暗号化領域の共有を求めるクライアントプロセスBは、乱数発生器によって生成されたチャレンジ乱数Bnと、公共機関により作成された証明書BcertをオーナープロセスAに送り、認証を要求する。

【0113】ここで、証明書Bcertは、公共機関が秘密鍵Ksqを使って作成した証明書である。公共機関によって発行されるクライアントプロセスBの公開鍵Kpbが埋め込まれており、公共機関の公開鍵Kpqを使うことによって公開鍵Kpbを抽出できる。

【0114】オーナープロセスAへの証明書の送信は暗号化される必要はなく、OSやライブラリによって提供されるシステムコールやAPIなどを利用してよい。これ以後、プロセス間で証明書などのデータを送受信するための手段は、特に説明しない限り、通常のOSやライブラリのサービスルーチンを利用することができる。

【0115】オーナープロセスAは、受け取った証明書Bcertの完全性の検証を公共機関の公開鍵Kpqを使って行う。検証の結果、公共機関による署名が正当であると判明した場合、オーナープロセスAは、鍵交換第1フェーズ値Avを計算する。このとき、クライアントプロセスBの公開鍵Kpbも求める。同時に、オーナープロセスAは、チャレンジ乱数Anと証明書AcertをクライアントプロセスBに送り、認証を要求する。クライアントプロセスBは、受け取った証明書Acertの完全性を検証し、公共機関による署名が正当であると判明した場合、鍵交換第1フェーズ値Bvを計算する。このとき、オーナープロセスAの公開鍵Kpaも求める。

【0116】オーナープロセスAは、鍵交換第1フェーズ値Avと、プロセスBのチャレンジ乱数Bnに対するメッセージ署名を作成し、クライアントプロセスBへ送る。

【0117】クライアントプロセスBも同様に、鍵交換第1フェーズ値Bvと、プロセスAのチャレンジ乱数Anに対するメッセージ署名を作成し、オーナープロセスAへ送る。

【0118】オーナープロセスAは、クライアントプロセスBの公開鍵Kpbを用いて、クライアントプロセスBから受け取ったメッセージ署名を調べ、このメッセージ署名が変更されていないことを検証する。

【0119】オーナープロセスAは、乱数と、検証されたフェーズ値Bvを用いて、プロセスA、B間の認証された共通鍵Kab(あるいはKauth)を算出する。

【0120】クライアントプロセスBも同様に、オーナープロセスAの公開鍵Kpaを用いて、オーナープロセスAから受け取ったメッセージ署名を調べ、このメッセージ署名が変更されていないことを検証する。

【0121】クライアントプロセスBは、乱数と、検証されたフェーズ値Avを用いて、プロセスA、B間の認証された共通鍵Kab(あるいはKauth)を算出する。

【0122】このような認証方法を使用することによって、2つのプロセス間で、非暗号化通信路上で鍵情報を交換する場合に、悪意のあるプログラム(偽造OSも含む)が2プロセス間に介在して、認証鍵を改ざんしたり鍵自体を破る事態を防止することができる。

【0123】続いて、オーナープロセスAは、図5（第1実施形態）のステップS86と同様に、共通鍵K_{ab}（またはK_{auth}）を知っているプロセスA、Bからのみアクセスされる共有暗号化データ領域S_{ab}を作成する。共有暗号化データ領域S_{ab}は、ライブラリやOSが提供するメモリ割り当て手段を利用して作成される。領域S_{ab}の開始アドレス、領域サイズ（または終端アドレス）、属性値などは、OSの記憶リソース管理下に置かれる。共通鍵K_{ab}（またはK_{auth}）は、領域アドレスと関連付けて、暗号化属性レジスタに設定される。

【0124】次に、クライアントプロセスB、図5（第1実施形態）のステップS96と同様に、オーナープロセスAが作成した共有暗号化データ領域S_{ab}を、OSの提供するメモリ共有手段を利用して、自身のプロセス空間にマップする。このメモリ共有情報のやりとりは、暗号化されている必要は無い。プロセスBに関しても、プロセスBの領域S_{ab}用の暗号化属性レジスタに、共通鍵K_{ab}（またはK_{auth}）を含む情報を設定しておく。

【0125】これ以降、プロセスA、Bともに、共有暗号化データ領域S_{ab}へのアクセスは、マイクロプロセッサ（耐タンパプロセッサ）1の暗号化属性レジスタを介してアクセスされるため、共通鍵K_{ab}（またはK_{auth}）を知っているプロセス間でのみ安全に行うことができる。

【0126】＜第3実施形態＞第1および第2実施形態では、2プロセス間で共有暗号化データ領域を作成する手順について述べた。第3実施形態では、3以上のプロセスが、これらの間で共有暗号化データ領域H_{share}を構築する手順について説明する。

【0127】図7は、プロセスA、B、Cの各アドレス空間と、これらのプロセスが共有する共有暗号化データ領域の構成例を示す。図7の例では、プロセスAが共有暗号化データ領域H_{share}の生成者となるオーナープロセスである。図8は、共有暗号化データ領域の作成のための各プロセスの暗号化属性レジスタの構成例を示す。

【0128】まず、オーナープロセスAは、暗号化命令実行モードに移行し、自身のアドレス空間に共有されるべき暗号化データ領域H_{share}106を生成する。領域H_{share}用の鍵K_{share}は、あらかじめプログラムコード内に埋め込まれていてもよいし、あるいは、乱数発生器などを用いて初期値を生成し、これを利用して生成した鍵としてもよい。鍵K_{share}を動的に生成した場合には、この鍵を、秘匿された暗号化領域に保存する。共有暗号化データ領域H_{share}106のアドレスと、この領域のための鍵K_{share}は、マイクロプロセッサ1の内部の暗号化属性レジスタ216に設定される。

【0129】次に、オーナープロセスAは、第1実施形態で説明した方式に従い、プロセスBとの間でのみ共有される共有暗号化データ領域S_{ab}104を自己のアドレス空間に作成する。さらに、プロセスAとプロセスCとの

間で共有される共有暗号化データ領域S_{ac}105を作成する。すなわち、オーナープロセスAを中心として、暗号化データ領域を共有したい各プロセスとの間で1対1で共有される共有暗号化データ領域S_{ax}を作成する。図7の例におけるオーナープロセスAの共有領域S_{ab}、S_{ac}のアドレスと、これらの領域にアクセスするための共通鍵K_{ab}、K_{ac}は、それぞれ関連付けられて暗号化属性レジスタ214、215に設定される。

【0130】オーナープロセスAと、各クライアントプロセスとの間の1対1対応の共有暗号化データ領域S_{ax}は、プロセス間で最終的な共有領域となるH_{share}の鍵K_{share}を安全に通知する通信路として機能する。この意味で、2プロセス間の共有暗号化領域S_{ax}は、鍵通知領域と称されてもよい。具体的には、プロセスAは、プロセスBとの共有領域S_{ab}上に共有暗号化データ領域H_{share}用の鍵K_{share}を書き込む。プロセスBは、自己の空間に仮想的に共有暗号化データ領域S_{ab}を作成し、領域S_{ab}—S_{ab}間を通信路として鍵K_{share}を読み込む。これにより、プロセスAとBとの間で領域H_{share}の鍵K_{share}を共有することができる。

【0131】プロセスBはさらに、OSによって提供される通常の共有メモリと同様のメモリ共有手段を利用して、オーナープロセスAの共有暗号化領域H_{share}を、自己のプロセス空間にマッピングする。これにより、共有暗号化データ領域H_{share}を仮想的に自己のアドレス空間に作成する。プロセスBの領域S_{ab}および最終的な共有領域H_{share}のアドレスと、これらの領域用の鍵K_{ab}およびK_{share}は、それぞれ関連付けられて、プロセスBの暗号化属性レジスタ224、225にそれぞれ設定される。

【0132】同様に、プロセスCがプロセスAが作成した共有領域H_{share}を共有する場合には、オーナープロセスAによって領域S_{ac}上に書き込まれたH_{share}用の鍵K_{share}を、通信路S_{ac}—S_{ac}を介してプロセスCが読み込み、プロセスAの領域H_{share}を自己のプロセス空間へのマッピングすることで、領域H_{share}を共有することができる。鍵通知領域S_{ac}および共有暗号化データ領域H_{share}のアドレスと、鍵K_{ac}およびK_{share}は、それぞれ関連付けられ、プロセスCの暗号化属性レジスタ234、235に設定される。

【0133】これ以外のプロセスが共有暗号化データ領域H_{share}を共有する場合も、プロセスBやCと同様のシーケンスを繰り返すことで、オーナープロセスAとの間に安全な通信路が確立され、メモリ共有が可能となる。

【0134】なお、図7に示される各プロセスの暗号化実行コード/データ領域101、121、131や、秘匿データ領域H_a102、H_b123、H_c133の生成については、第1実施形態と同様であるので、説明を省略する。

【0135】第3実施形態では、3つのプロセス間でひ

とつの暗号化されたデータ領域を共有する例を説明したが、4以上のプロセス間でも同様の方法で暗号化データ領域を共有することができる。この場合、共有プロセスの数を n 個とすると、オーナープロセスとの間に、 $n-1$ の通信路、すなわち共有領域 S_{ax} を形成し、この通信路を介して共通鍵 K_{share} の通知を行なうことによって、これらのプロセス間でのみ暗号化データ領域 H_{share} を共有することが可能になる。

【0136】

【発明の効果】以上説明したように、暗号化データ領域を2つのプロセス間で共有するために、各プロセスは暗号化命令実行モードに移行してから鍵交換を行なう。共通鍵と、その計算に使用された一時データは、各プロセスのみが読める秘匿データ領域またはプロセッサ内部のレジスタ上に置かれるので、OSの特権からも保護された安全な鍵共有が可能になる。

【0137】また、安全な鍵交換の結果生成された共通鍵に基づき、一方のプロセスが共有されるべき暗号化データ領域を作成し、他方のプロセスはそれを自己のアドレス空間にマップする。マイクロプロセッサ内部の暗号化属性レジスタに、各プロセスの共有暗号化データ領域と共通鍵とを関連付けて設定することにより、暗号化データ領域の共有が安全に行なわれる。

【0138】さらに、3以上のプロセス間でひとつの暗号化データ領域を共有する場合は、オーナープロセスと他の任意のプロセスとの間で共有される共有暗号化領域を、鍵通知のための安全な通信路として利用することにより、3以上の正当なプロセス間での最終的な共有暗号化データ領域の共有が安全に行なわれる。

【図面の簡単な説明】

【図1】本発明によるプロセス間でのデータ共有方法の前提となるマイクロプロセッサの構成図である。

【図2】図1のマイクロプロセッサ内部の暗号化属性レジスタと鍵格納データの構成例を示す図である。

10

【図3】プロセス内で生成される暗号化領域と、領域の生成に用いる鍵と、これらに対応付けて格納する暗号化属性レジスタの関係を示す図である。

【図4】本発明の第1実施形態における2つのプロセス間での暗号化データ領域の共有の模式図と、対応する暗号化属性レジスタとを示す図である。

【図5】図4に示す2つのプロセス間での暗号化データ領域の共有を行なうための処理手順を示すフローチャートである。

【図6】本発明の第2実施形態における、2つのプロセス間でのデータ共有をさらに安全に行なうための認証手続きを示すシーケンス図である。

【図7】本発明の第3実施形態における、3つのプロセス間での暗号化データ領域の共有を示す模式図である。

【図8】図7に示す3プロセス間での暗号化データ領域共有を実現するための、各プロセスの暗号化属性レジスタの構成例を示す図である。

【符号の説明】

1 マイクロプロセッサ

2 命令TLB

3 プロセッサコア

4 命令実行部

5 1次キャッシュ

6 コード・データ暗号化/複号化部

7 バスインターフェースユニット

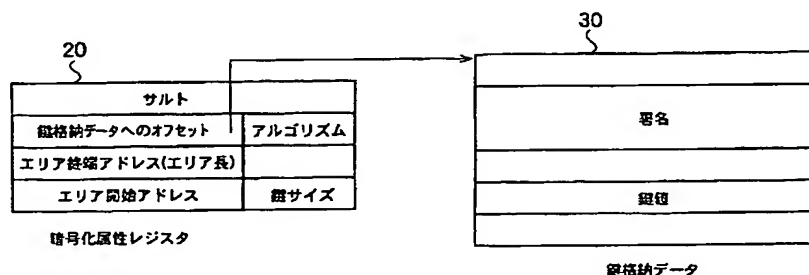
20、51、71、211、221、231 暗号化属性レジスタ

44、64、103、123、133 秘匿データ領域 H_x

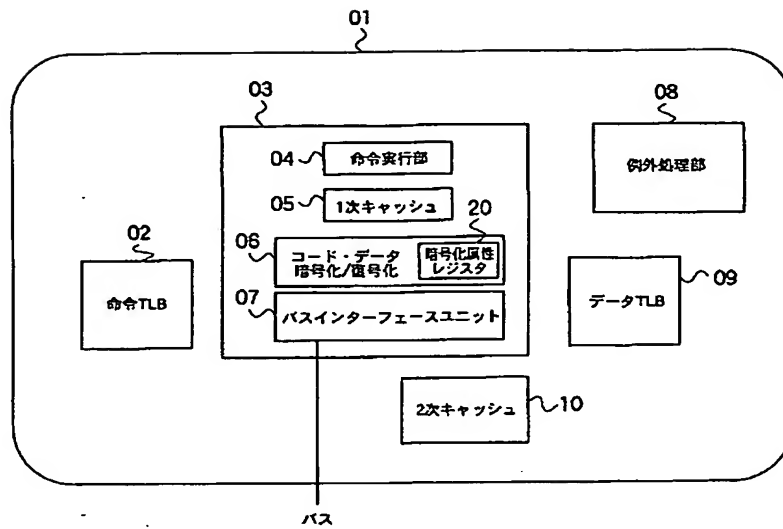
30 45、64、104、105、124、134 共有暗号化データ領域 S_{ax}

106、125、135 共有暗号化データ領域 H_{share}

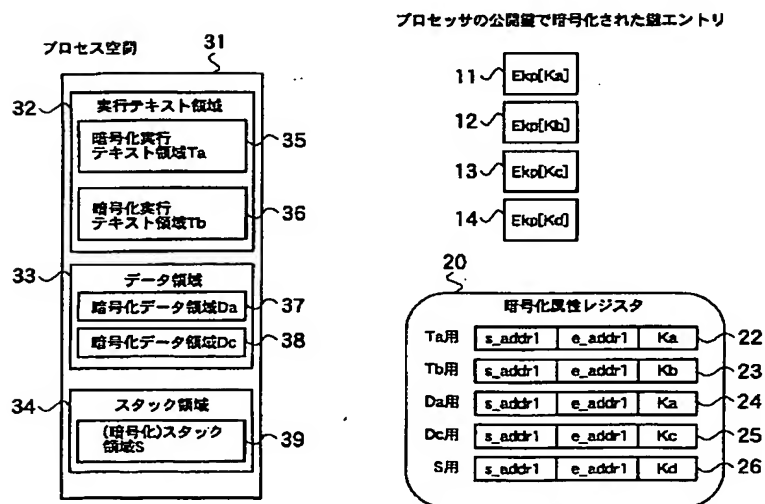
【図2】



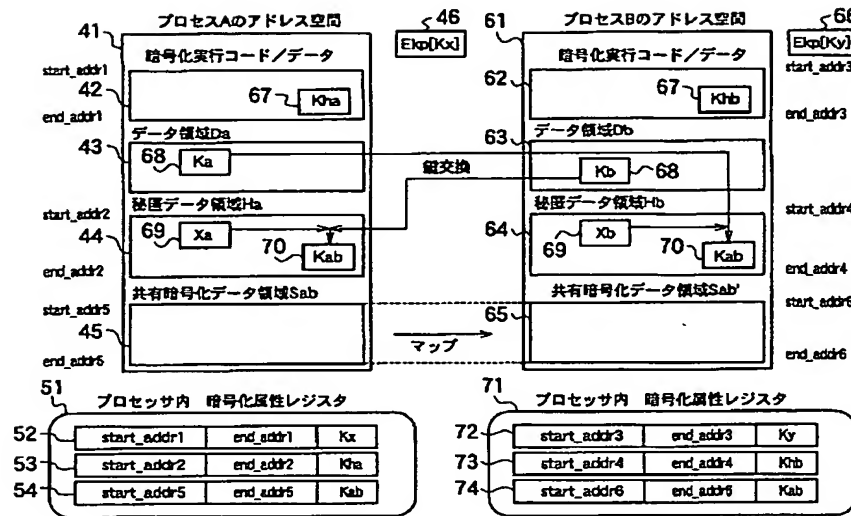
【図1】



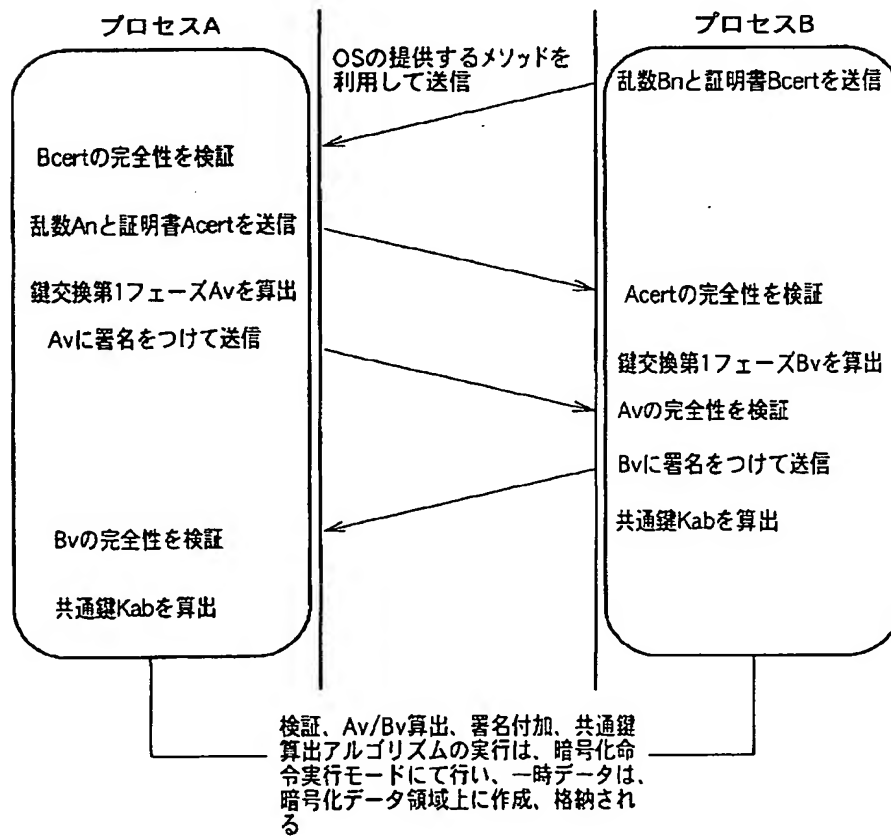
【図3】



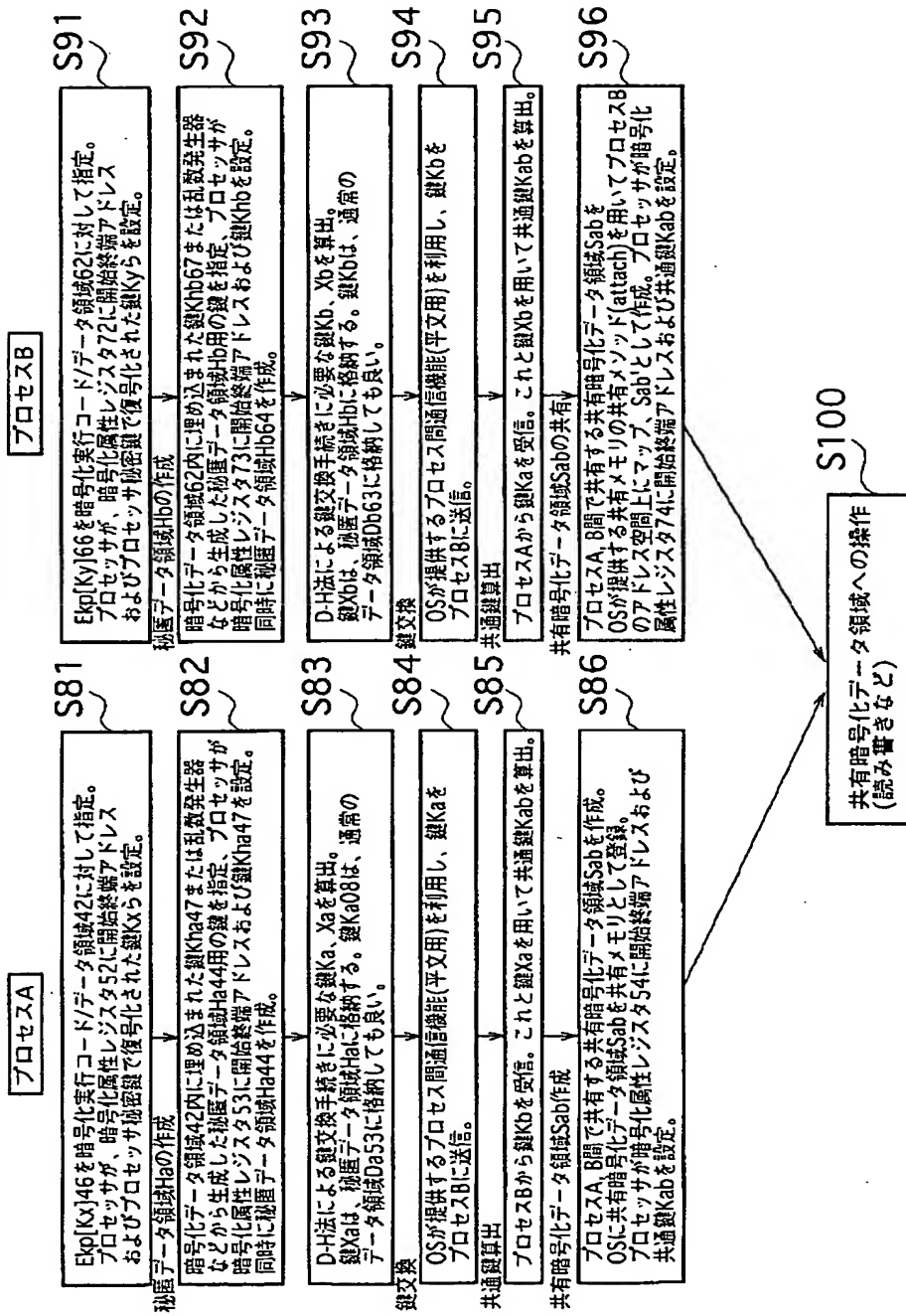
【図4】



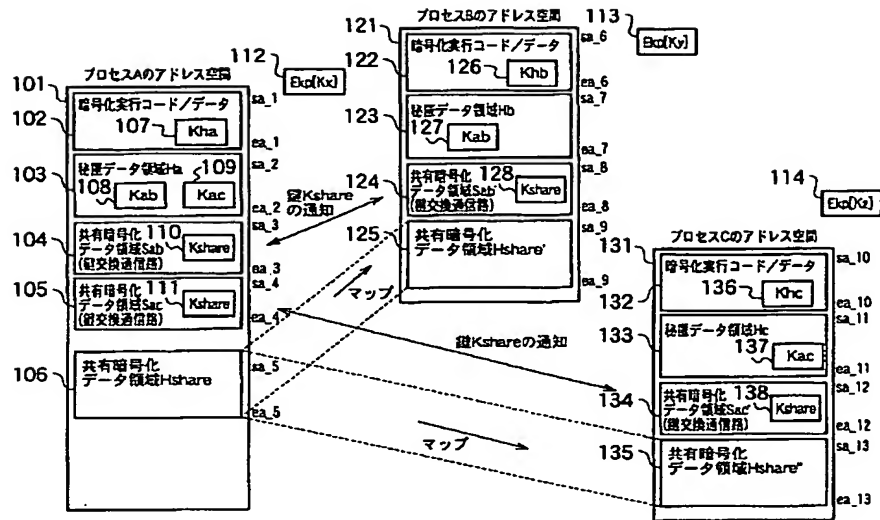
【図6】



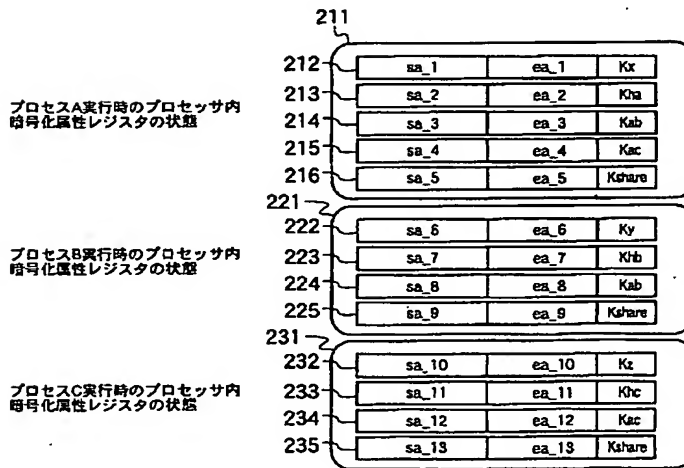
【図5】



【図7】



【図8】



フロントページの続き

(72)発明者 白川 健治
 神奈川県川崎市幸区小向東芝町1番地 株
 式会社東芝研究開発センター内
 (72)発明者 尾崎 哲
 神奈川県川崎市幸区小向東芝町1番地 株
 式会社東芝研究開発センター内

(72)発明者 藤本 謙作
 神奈川県川崎市幸区小向東芝町1番地 株
 式会社東芝研究開発センター内
 Fターム(参考) 5B017 AA03 BA07
 5B076 FA00
 5B098 GA04 JJ08
 5J104 AA16 EA02 EA04 EA24 NA02
 NA42

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.